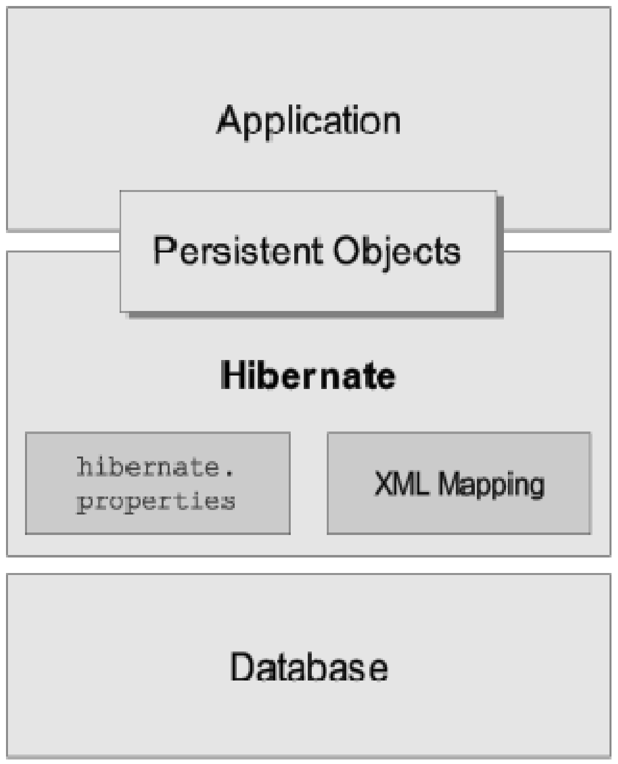
Annotations in

HIBERNATE

Remember the Structure



Hibernate Model

- Hibernate Core offers native API's & object/relational mapping with XML metadata
- Hibernate Annotations offers JDK 5.0 code annotations as a replacement or in addition to XML metadata
- Hibernate EntityManager involves standard JPA for Java SE and Java EE

JPA (Java Persistent API)

- JPA is a part of EJB 3.0 specification
- JPA is a POJO API for object/relational mapping that supports the use both of Java metadata annotations and/or XML metadata

What is Annotation ?

- Annotation is a specific construction in java 5 for adding additional information to Java source code
- Annotations are embedded in class files generated by compiler & can be used by other frameworks

Annotation Using Syntax

@AnnotationName (element1 = “value1”, element2 = “value2”)

@AnnotationName (“value”)

Can be used for

- classes
- methods
- variables
- parameters
- packages
- annotations

Hibernate Annotations

- Basic annotations that implement the JPA standard
- Hibernate extension annotations

Annotated Java class

```
@Entity                                          // Declares this an entity bean
@Table(name = "people")                          // Maps the bean to SQL table "people"
class Person implements Serializable{

  @Id                                            // Map this to the primary key column.
  @GeneratedValue(strategy = GenerationType.AUTO) // Database will generate new primary keys
  private Integer id;

  @Column(length = 32)                           // Truncate column values to 32 characters.
  private String name;

  public Integer getId() {
    return id;
  }

  public void setId(Integer id) {
    this.id = id;
  }

  public String getName() {
    return name;
  }

  public void setName(String name) {
    this.name = name;
  }
}
```

Basic Annotations

• @Entity	Declares this an entity bean
• @Id • @EmbeddedId • @GeneratedValue	Identity
• @Table • @Column	Database Schema Attributes
• @OneToOne • @ManyToOne • @OneToMany	Relationship mappings

& etc.

Extension Annotations

- Contained in org.hibernate.annotations package
- Examples:

    @org.hibernate.annotations.Entity

    @org.hibernate.annotations.Table

    @BatchSize

    @Where

    @Check

    & .etc

hibernate.cfg.xml

```
<hibernate-configuration>
        <session-factory>
          <property
   name="hibernate.connection.driver_class">org.hsqldb.jdbcDriver</property>
          <property name="hibernate.connection.url">jdbc:hsqldb:hsql://localhost/
  </property>
          <property name="hibernate.connection.username">sa</property>

          <property
   name="dialect">org.hibernate.dialect.HSQLDialect</property>


          <mapping class="hello.Person"/>             --------   !!!  ----------
        </session-factory>
   </hibernate-configuration>
```

Maven – pom.xml - dependencies

```
<dependency>
   <groupId>org.hibernate</groupId>
   <artifactId>hibernate-core</artifactId>
   <version>3.6.8.Final</version>
</dependency>
```

- Easier compared to version 3.3.x

Maven – pom.xml - repositories

```xml
<repository>
      <id>jboss-public-repository-group</id>
      <name>JBoss Public Maven Repository Group</name>
      <url>https://repository.jboss.org/nexus/content/groups/public-jboss/</url>
      <layout>default</layout>
      <releases>
            <enabled>true</enabled>
            <updatePolicy>never</updatePolicy>
      </releases>
      <snapshots>
            <enabled>true</enabled>
            <updatePolicy>never</updatePolicy>
      </snapshots>
</repository>
```

Hibernate Class Entities

- Class attributes
  - Hibernate uses reflection to populate
  - Can be private or whatever
- Class requirements
  - Default constructor (private or whatever)
    - “However, package or public visibility is required for runtime proxy generation and efficient data retrieval without bytecode instrumentation.”
- JavaBean pattern common
  - Not required though but easier
- 3 methods of serialization definition
  - Following slides

Hibernate Annotation Mappings

- Annotations in code
  - Beginning of class
  - Indicate class is Entity
    - Class doesn’t have to implement java.lang.Serializable
  - Define database table
  - Define which attributes to map to columns
    - Supports auto-increment IDs too
    - Can dictate value restrictions (not null, etc)
    - Can dictate value storage type
- Existed before JPA standard (later slides)
- Doesn’t require a separate hbm.xml mapping file (discussed later)
  - But is tied to code

Hibernate Annotation Example

```
@Entity
@Table( name = "EVENTS" )
public class Event {
   private Long id;
…
     @Id
     @GeneratedValue(generator="increment”
     @GenericGenerator(name="increment", strategy = "increment")
     public Long getId() {  return id;    }

     @Temporal(TemporalType.TIMESTAMP)
     @Column(name = "EVENT_DATE")
     public Date getDate() { return date;    }
     public void setDate(Date date) { this.date = date;    }
…
}
```

Hibernate Annota

Tells hibernate this goes into the EVENTS table

```
@Entity
@Table( name = "EVENTS" )
public class Event {
   private Long id;
…
     @Id
     @GeneratedValue(generator="increment”
     @GenericGenerator(name="increment", strategy = "increment")
     public Long getId() {  return id;    }

     @Temporal(TemporalType.TIMESTAMP)
     @Column(name = "EVENT_DATE")
     public Date getDate() { return date;    }
     public void setDate(Date date) { this.date = date;    }
…
}
```

Hibernate Annota[illegible]

Tells hibernate that this is an auto generated field for the database

```
@Entity
@Table( name = "EVEN[illegible])
public class Ev[illegible]nt {
   private Long id;
…
     @Id
     @GeneratedValue(generator="increment”
     @GenericGenerator(name="increment", strategy = "increment")
     public Long getId() {  return id;    }

     @Temporal(TemporalType.TIMESTAMP)
     @Column(name = "EVENT_DATE")
     public Date getDate() { return date;    }
     public void setDate(Date date) { this.date = date;    }
…
}
```

Hibernate Annota

Note that you don’t need any annotations on the actual private fields or setters if you use the standard JavaBean pattern. The getter defines it.

```
@Entity
@Table( name = "EVENTS" )
public class Event {
   private Long id;
…
      @Id
      @GeneratedValue(generator="increment”
      @GenericGenerator(name="increment", strategy = '       nent")
      public Long getId() {  return id;    }

      @Temporal(TemporalType.TIMESTAMP)
      @Column(name = "EVENT_DATE")
      public Date getDate() { return date;    }

      public void setDate(Date date) { this.date = date;    }
…
}
```

Hibernate Annota

Also note that this is automatically stored with a column name of “title” so we didn’t have to add any annotations

```
@Entity
@Table( name = "EVENTS" )
public class Event {
…
    private String title;

    public String getTitle() {      return title;    }

    public void setTitle(String title) {     this.title = title;     }
…
}
```

JPA Annotation

- Became standard
  - Came after Hibernate annotations
- Works almost like Hibernate annotations
  - Requires “META-INF/persistence.xml” file
    - Defines data source configuration
    - HibernatePersistence provider
      - Auto-detects any annotated classes
      - Auto-detects any hbm.xml class mapping files
        - (later slides)
      - Allows explicit class loading for mapping
- Annotation syntax
  - Same as Hibernate
  - Hibernate has a few extensions (see docs)